System Requirements Specification Index

For

Wealth Management Advisory System

Version 1.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Wealth Management Advisory System
System Requirements Specification

---

1. Business-Requirement:

1.1 Problem Statement:

Wealth Management Advisory System is .Net Core web API 3.1 application integrated with MS SQL Server, where it refers to development of a Wealth Management Advisory System that provides users with personalized financial advice, investment tracking, and portfolio management tools. As individuals seek to grow their wealth through investments, they need an integrated platform that offers tailored advice, tracks investments, and manages their portfolios. This system will enable users to understand their financial health, explore investment options, and receive guidance on achieving their financial goals.

1.2 Following is the requirement specification:

	Wealth Management Advisory System
Modules	
1	User
2	Investment
3	Portfolio
4	Advisory
Functionalities	
1	Get All User Details
2	Create User
3	Retrieve User by id
4	Update User
5	Delete User
6	Add Investment
7	Get Investment
8	Add Portfolio
9	View Portfolio
10	Rebalance Portfolio
11	Get Investment Recommendation
12	Schedule Advisory Session
13	Get Advisory Details

2. Assumptions, Dependencies, Risks / Constraints

2.1 Wealth Management Advisory Constraints:

- If any Id does not exist then the operation should throw a custom exception.
- While fetching the any details by id, if id does not exist then the operation should throw a custom exception.

2.4 Common Constraints

- For all rest endpoints receiving [FromBody], validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in model classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3. Business Validations

3.1 User Class Entities

- **UserId** (int, primary key): Unique identifier for the user.
- **Income** (decimal): The current income of the user.
- **Risk Appetite** (String): Describes the risk tolerance of the user.
- **Financial Goals** (string): Describes the financial goals set by the user.

3.2 Investment Class Entities

- **InvestmentId** (int, primary key): Unique identifier for the investment.
- **UserId** (int): Unique identifier of the user associated with the investment.
- **Amount** (decimal): The amount to be paid in the investment transaction.
- **Investment Type** (string): The type of the investment (e.g. Gold, MF).
- **PurchaseDate** (DateTime): The date of investment starts.

3.3 Portfolio Class Entities

- **PortfolioId** (int, primary key): Unique identifier for the portfolio.
- **UserId** (int): The user associated with the portfolio.
- **Total Investment Value** (decimal): The amount of money involved in the investment.

- **Portfolio Performance** (decimal): The amount of money gained in the investment.
- **Asset Allocation** (string): The assets available per user.

3.4 Advisory Class Entities

- **AdvisoryId** (int, primary key): Unique identifier for the advisory.
- **UserId** (int): The unique identifier for the user who triggered the event.
- **Recommendation Title** (string): The type of recommendation.
- **Recommendation Description** (string): A description or details about the advisory.
- **Advisory Date** (DateTime): The date and time the event occurred.
- **Session Type** (string): The type of session.
- **Risk Level** (string): level of risk.

3.5 Response Class Entities

- **Status** (string): Shows success/error message.
- **Message** (string): Shows description.

4. Considerations

- You can perform the following possible actions

User,Investment,Portfolio,Advisory

Rest Endpoints

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 UserController

URL Exposed		Purpose
/api/user/all Http Method	GET	

Parameter 1	-
Return	List<User>

Get All user Details

/api/user/create

Http Method	POST
Parameter 1	User model
Return	Response Entity

Create user

/api/user/{userId}

Http Method	GET
Parameter 1	Int userId
Return	User

Retrieve user by id

/api/user/update/{userId}

Http Method	PUT
Parameter 1	Int userId
Parameter 2	User model
Return	Response Entity

Update user

/api/user/delete/{userId}

Http Method	DELETE
Parameter 1	Int userId
Return	Response Entity

Delete user

5.2 InvestmentController

URL Exposed	Purpose
/api/investment/add Http Method: POST Parameter 1: Investment model Return: Response Entity	Add Investment
/api/investment/{userId} Http Method: GET Parameter 1: Int userId Return: User	Get Investment by user id

5.3 PortfolioController

URL Exposed	Purpose
/api/portfolio/add Http Method: POST Parameter 1: Portfolio model Return: Response Entity	Add Portfolio
/api/portfolio/view/{userId} Http Method: GET Parameter 1: Int userId Return: Portfolio	Get portfolio by id
/api/portfolio/rebalance/{userId} Http Method: PUT Parameter 1: Int userId Parameter 2: Portfolio model Return: Portfolio	Rebalance Portfolio

5.4 AdvisoryController

URL Exposed	Purpose
/api/advisory/recommendations/{userId} Http Method: GET Parameter 1: Int userId Return: Advisory	Get advisory recommendations
/api/advisory/schedule Http Method: POST Parameter 1: Advisory model Return: Response Entity	Schedule Advsiory
/api/advisory/details/{advisoryId} Http Method: GET	Get Advisory Details By Id

Parameter 1	Int advisoryId	
Return	Advisory	

6. Template Code Structure

6.1 Package: WealthManagementAdvisory

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	UserController, InvestmentController, PortfolioController, AdvisoryController	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: WealthManagementAdvisory.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			

Interface	IUserService,IInvestmentService,IPortfolioService,IAdvisoryService interface	Inside all these interface files contains all business validation logic functions.	Already implemented
Service	UserService,InvestmentService,PortfolioService,AdvisoryService CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially implemented
Repository	IUserRepository,InvestmentRepository,IPortfolioRepository,IAdvisoryRepository (CS files and interfaces)	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially implemented

6.3 Package: WealthManagementAdvisory.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	WealthMangamentDBContext cs file	All database Connection,collection setting class	Already Implemented

6.4 Package: WealthManagementAdvisory.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	User ,Investment,Portfolio,Advisory (CS files)	All Entities/Domain attribute are used for pass the data in controller and status entity to return response Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Partially implemented

7. Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (WealthManagementAdvisory /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**
     1> Create Database WealthManagementDb
     2> Go

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (WealthManagementAdvisory /dotnet-ef database update)

6. To check whether migrations are applied from terminal:
   (WealthManagementAdvisory /sqlcmd -S localhost -U sa -P pass@word1)

```
1> Use WealthManagementDb
2> Go
1> Select * From __EFMigrationsHistory
2> Go
```

7. To build your project use command:
   (WealthManagementAdvisory /dotnet build)

8. To launch your application, Run the following command to run the application:
   (WealthManagementAdvisory /dotnet run)

9. This editor Auto Saves the code.

10. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.

11. To test web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

    Note: The application will not run in the local browser

12. To run the test cases in CMD, Run the following command to test the application:
    (WealthManagementAdvisory.Tests/dotnet test --logger "console;verbosity=detailed")
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

14. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

15. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**
